INFO 290 SPRING 2023 - Climate, People, and Informatics

Final Paper - Calculating the Cost Savings Associated with an Industrial HVAC System

Erin Jones - 5/10/2023

Proposed Idea:

The stated goal at the start of this project was ambitious. I set out to carve a path, using machine learning, for calculating cost savings based on real-world HVAC data. The highly ambitious nature increased the likelihood I would meet my true goal: to learn a bit more about the following topics in preparation for my summer internship, while working with real-word data:

- creating an aggregate time series by combining two sets of time series with different features
- practicing data manipulation with pandas + other big data packages in python
- learn more about forecasted time series + removing seasonality from data
- understanding electricity pricing
- deploying a supervised learning algorithm + learn more about machine learning algorithms in general

Background Information:

The data I chose to work with was produced by a single site upon which PowerTron had completed a "treatment". The treatment occurs when a nanotechnology is injected into an HVAC device, coating the inside of the device with the hopes of returning the device to manufactured state or better in terms of device efficiency.

A sensor system is attached to the device several months prior to treatment, producing several months of minute-by-minute time series data that represent baseline behavior for the device. After treatment occurs, the sensor continues to capture data for months after with the goal of being able to track how the device is functioning when compared to baseline. Neither period, pre or post-treatment, contains data representing a full year of device behavior.

The 37 variables/features collected include: `tags, time, velocity, cfm, compressor1, compressor2, compressor3, compressor4, COP, Delta_Enthalpy, deltat, dewpoint1, dewpoint2, eer, kw, kwton, currenta, currentavg, currentb, currentc, powersum, VoltAB, VoltBC, VoltAC, VoltAvg, totalCapacity_TONS, OAtemp, powerfactor, humidity1, temp1, enthalpy1, humidity2, temp2, enthalpy2, totalCapacity_BTU, wetbulb1, wetbulb2.`

Several features are derived/constructed features that are calculated for each measurement based on other features. These include cfm, COP, delta_enthalpy, deltat, eer, kw, kwton, currentavg, powersum, totalcapacity_tons, powerfactor, and totalCapacity_BTU. It is important to note these features, as if a correlation matrix or heat map are created, they will automatically produce strong correlations to their component variables (e.g. currentavg to kw, kwton, currenta, currentb, currentc, and powersum and vice versa) which is a result of the function-based origin of these constructed features as opposed to a truly strong correlation between raw measurements. Treatment occured July 6th at 17:30 for this unit. There are technically two units on site for this location, however I chose to begin with a single unit (458) for the purposes of this investigation.

With HVAC and any other dataset produced by an engineering phenomenon, there is going to be a fair amount of expected seasonality in the time series -- points are unlikely to be distributed randomly. With residential HVAC, this is especially the case, as temperatures change in a non-random fashion when considering temperature trends throughout the year. In the US, it is likely to be warmer in the summer and cooler in the winter and the hotter it is, the likelihood of an HVAC device being used increases. This means that without having a year of data representing baseline functionality (pre-treatment) to compare directly with a year of data representing post-treatment device functionality, I needed to find a way to forecast the time series so I can compare apples to apples (e.g. February pre-treatment vs February post-treatment) to come up with a quantity for electicity savings, which is based on consumption.

The site being used for initial exploration of the solution space is produced by an HVAC device associated with a cell tower site in Fresno, CA. Residents and commercial clients are serviced by PG&E in Fresno, CA [1]. Unfortunately, PowerTron did not have access to any additional data about client electricity plans.

Based on the above information, I needed to dig deeper into the following:

- How can I calculate electricity consumption based on the time series given (kW to kWh)?
- Is it possible to find historical data on month to month price per kWh for electricity from PG&E?
- What kind of models can be used to create fake/simulated time series to overcome seasonality instead of using traditional statistical methods?
- Are there any challenges or limitations to using machine learning models in this capacity?
- Will I stumble on any leverage points or information problems while exploring this real world data set?

Electricity Rates

In starting this project, I had imagined I could find some sort of historical database on PG&E's website showing the month-by-month price per kWh used, which could then be multiplied by the number of kilowatthours for the month to come up with the total cost to the user for that month. After some digging, I found a total of 107 different tariffs offered by PG&E for electricity [3]. These tariffs were segmented by a combination of features such as the general nature of the client (residential, small to medium sized business, and large business), the specific client type (agricultural, government, direct access), and finally the PG&E-specialized agreements that the utility uses to incentivize electricity customers to do things such as install solar panels and feed excess energy back to the grid, buy plug-in EVs, customer owned street and highway lighting, and the list goes on.

There seems to be an information problem here, as I had to wade through several pages on their sitemap to even find the tariffs index cited above. In fact, I found several pages that I believed would send me to historical rate data, but instead provided me with a history of advice filings related to tariffs for the given year which was found under tariffs > data [4]. At pretty much every turn, PG&E urges its customers to work with the utility to 'find the right plan for them' and 'maximize their savings' by working with an agent. As someone looking for data, this solution felt disatisfied, but as I began to learn about the various rates and incentive plans offered by the company, I quickly became lost in the weeds. In terms of leverage points, while PG&E is trying to do right by the public and share its data, it is truly a disincentive falling between the rules of the system and information flows [5]. Instead of creating a site and investing in the user experience and navigability of this experience, they seemingly dumped a bunch of information out there and then ceded to the fact that it was confusing, so they offer a 'helping hand'. The biggest issue with this approach is that for a customer, this type of 'contact us' approach may be genuine, but it can still wreak of a company

just trying to make another sale. PG&E maintains the power in this situation and the average citizen (be they a residential or business client) is offered only an illusion of transparency. The leverage points of information flows and incentives is feeble when there is no change in the true distribution of power over the system, which is higher in the hierarchy of leverage points offered by Meadows.

For the information flows to create the leverage of clients making informed decisions about their energy bills, this information needs to be presented in a manner that allows them understanding and thereby autonomy and control over the information presented. This could be completed in the form of a better organized site and more information resources about plan types that can be engaged with independent of PG&E. Offering an even higher level of impact would be to create some sort of simulation of what different plans would produce in different scenarios that allows users a tangible understanding of what the plan could produce. Maybe the user could input some information about themselves (e.g. I am a small business owner, I currently have this type of plan, I have these distributed energy resources associated with my site, etc.) and then allow them to shift their use and make changes that would show a theoretical forecast based on several plans. PG&E does have a module where current clients can elect for PG&E to analyze their rates and will optimize/select the best plan for them. Again, this comes across in a black-box manner that screams 'trust us' and also looks only backwards at a user's consumption. Having something foreward looking that allows users to participate and modify features to spit out forecasted scenarios would be an amazing feature to offer that may provide greater levels of user agency and autonomy - targeting the overall rules of the system in a way that is meaningful to users. This platform would also allow for greater flexibility for those individuals living in multi-family communities that do not have direct control over the plans offered by their communities.

It might be an interesting research project in the future to set up some sort of experiment that investigates the PG&E line or interviews PG&E clients across the spectrum about their experiences interacting with the 'plan advice' service offered by PG&E. A similar deep dive could be completed on PG&E's plan selection optimization tool, as it could not be investigated without multiple plan types and sites to better understand the experience offered by PG&E.

Returning to the conundrum of sourcing rate data for our cell tower client, I hit a dead end. While I could rule out certain tariffs, such as residential or agricultural plans, from the list of hundreds, I could not trim it down to less than five or so plans that might work for the client. I don't know if the cell site was making use of distributed enegry resources like solar panels, for which PG&E might provide special tariffs or while could simply reduce the load for which the client is charged altogether. It is possible that this simulation could be run for multiple scenarios based on plan if given more time, but it would not be a matter of simply modifying the rate. PG&E offers time-of-use pricing and peak day pricing that would require special considerations when building a simulation. With more time, I could produce estimates for the maximum amount saved (using the most expensive rates) and the minimum amount saved (using least expensive rates) and build in several scenarios that account for time. These would be general ballpark estimates and could result in frustration from clients if the client does not properly understand the basis for these estimates and how modifications in time of consumption or plan or electricity rates might impact their true savings post-treatment.

To offer clients these types of insights, PowerTron would need to understand what tariff structure their client was currently engaged with in addition to understanding any DER capacity held by the client. This would require investment in a team that could create estimates based on generalized data (e.g. the client has solar

panels on site and has this type of rate plan from the utility), or the work of a singular individual who had access to more granular data, which is reliant on the comfort of clients sharing such data.

kW to kWh

On its face, this problem appears to be relatively simple. I had to do a little brushing up of my understandings of power, vs current, quantity of electricity consumed. "One kWh is one kilowatt generated or consumed for one hour" [2]. To calculate kWh, you typically would just multiply the rating of an electrical device (e.g. a 40-Watt lightbulb) by the number of hours that it is on. In this case, we do not have kWh, just kW and we do not have the rating of the machine.

Without knowing the rating, it is possible to estimate the rating by taking the average rating over the period of time the device is on. This is done below by subsetting the dataset to only the times where the device is active (the kw measurement is greater than zero) and then calculating the total average. This approach produced an average of ~ 0.026 kW for the entire pre-treatment dataset and then ~2.04 kW for the dataframe when masked to only include readings from when the device itself is on pre-treatment. For the post-treatment data, the kW readings are infintesimally large, reading thousands and thousands of kW. The results are incredibly odd given that a standard central HVAC is usually rated somewhere between 3 and 4 kW [6]. The pre-treatment data indicates a kW rating that is entirely too low and the post-treatment data indicates the opposite -- erroneously high. At first I was alarmed by this, thinking maybe the data was wrong or the sensor was out of whack. I racked my brain and remembered something PowerTron had told me - the device acts weird as it stabilizes post-treatment. We should see a stabilization several months after the treatment. Sure enough, when I plotted the kw for the active post-treatment data, there is a huge spike in the kW measurement around the time of treatment as the device tries to stabilize under the conditions of its coating. Several days out, the statistical summary appears to be much more normal and several months out, the device has settled into the range for what we might expect for a central ac unit [6].

The kW rating is a few kW higher post treatment. This is an interesting trend, as it would point to an HVAC bill that is quite a bit higher post treatment. The outdoor air temperature should result in more time spent on, but not any change in the kW being pulled by the device. It would be important in the future to take a closer look at this trend and further discuss with Powertron what may be causing it.

The fact that this period of fluctuation impacts the method of using the overall mean as a proxy for the kW rating for the device is slightly concerning. We would want to investigate further to pinpoint when the device is truly functioning 'normally' post treatment. This could be a cutoff dictated by PowerTron, or previous work on clustering may be of use to determine, based on the relationship between data points, how we might want to cull the data such that it wouldn't throw off a training dataset.

With the average data for each time series, one can sum the minutes where the kW rating is measured as above 0 and then divide by 60 to determine hours on. From there, you have kWh, which would need to be segmented monthly to produce a monthly value to multiply by a monthly tariff value which one could multiply the average kilowatt proxy rating for the appliance to derive a monthly electricity cost associated with the device running.

I did attempt several other methods of deriving kWh, including thresholding and measuring average kW for only time periods where the device was indicated to be on. This yielded unusable results in the form of a list

that were also not far off from the method of averaging. Due to limited time, I was unable to render this method more useful and therefore abandoned the results.

Due to the limitations discussed in accessing tariff data, generating a quantified cost savings was not possible at this time. However, this derivation of kWh and deeper understanding of sensor data when compared to electricity consumption will help me immensely in my summer internship. As a final weakness in my approach, I failed to work with BTUs or powersum as features that may have offered a less generalized and more fruitful approach to arriving at kWh.

In [1]:
```
# import necessary packages; pandas, matplotlib, and numpy are all native. Statsmodels, plotly ar
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf
import numpy as np
import plotly.graph_objects as go
import seaborn as sns
```

In [2]:
```
# combine csv to produce one df for pre-treatment and one df for post-treatement
df_pre = pd.concat(map(pd.read_csv, ['2021_0623_0630.csv', '2021_0701_0708.csv']))
df_post = pd.concat(map(pd.read_csv, ['2021_0701_0708.csv','2021_0708_0715.csv','2021_0715_0722.(
                                      '2021_0722_0730.csv','2021_0801_0808.csv','2021_0815_0822.(
                                      '2021_0901_0908.csv','2021_0908_0915.csv','2021_0915_0922.(
                                      '2021_1001_1008.csv','2021_1008_1015.csv','2021_1015_1022.(
                                      '2021_1101_1108.csv','2021_1108_1115.csv','2021_1115_1122.(
                                      '2021_1201_1208.csv','2021_1208_1215.csv','2021_1215_1222.(
                                      '2022_0101_0108.csv','2022_0115_0122.csv', '2022_0122_0131

# change datatype for datetime column
df_pre['time'] = pd.to_datetime(df_pre['time'])
df_post['time'] = pd.to_datetime(df_post['time'])

# Drops tags and name value
df_pre.drop(['tags', 'name'], axis=1, inplace=True)
df_post.drop(['tags', 'name'], axis=1, inplace=True)

# Trimming each DF to only include times before and after the treatment respectively
df_pre = df_pre[df_pre['time'] < '2021-07-06 17:30:00']
df_post = df_post[df_post['time'] > '2021-07-06 17:30:00']
```

In [3]:
```
# modify default pandas dataframe column display option + print out the dataframe
pd.set_option('display.max_columns', None)
df_pre
# df_post
```

Out[3]:

	time	velocity	cfm	compressor1	compressor2	compressor3	compressor4	COP	Delta_E
0	2021-06-23 15:28:31	0.0	0.000	0	0	0	0	0.0	
1	2021-06-23 15:43:11	0.0	0.000	0	0	0	0	0.0	
2	2021-06-23 15:43:13	0.0	0.000	0	0	0	0	0.0	
3	2021-06-23 15:43:14	0.0	0.000	0	0	0	0	0.0	
4	2021-06-23 15:43:16	0.0	0.000	0	0	0	0	0.0	
...	...	...	...	...	...	...	...	...	
1211	2021-07-06 17:29:44	32681.5	21896.605	0	0	0	32518	0.0	
1212	2021-07-06 17:29:47	32681.5	21896.605	0	0	0	32518	0.0	
1213	2021-07-06 17:29:51	32681.5	21896.605	0	0	0	32518	0.0	
1214	2021-07-06 17:29:54	32681.5	21896.605	0	0	0	32518	0.0	
1215	2021-07-06 17:29:57	32681.5	21896.605	0	0	0	32518	0.0	

92087 rows × 36 columns

In [4]:

```
# create a boolean dummy variable to show when the device is on based on compressor functionality
df_pre['device_on'] = (df_pre['kw'] > 0).astype(int)
df_post['device_on'] = (df_post['kw'] > 0).astype(int)

# df_pre['device_on'] = ((df_pre['compressor1'] > 0) | (df_pre['compressor2'] > 0) | (df_pre['co
# df_post['device_on'] = ((df_post['compressor1'] > 0) | (df_post['compressor2'] > 0) | (df_post[

# filter the df based on the boolean dummy to create a timeseries that only contains intervals wh
active_df_pre = df_pre[df_pre['device_on'] == True]
active_df_pre = active_df_pre.set_index(active_df_pre['time'])
active_df_post = df_post[df_post['device_on'] == True]
active_df_post = active_df_post.set_index(active_df_post['time'])
```

In [5]:
```
# Take a look at the statistical summary between the dataset representing only
# time when the device is on vs the full dataset
print(df_pre['kw'].describe())
print(active_df_pre['kw'].describe())
print(df_post['kw'].describe())
print(active_df_post['kw'].describe())
```

```
count    92087.000000
mean         0.026200
std          0.229869
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          2.110000
Name: kw, dtype: float64
count    1181.000000
mean        2.042887
std         0.014650
min         2.000000
25%         2.030000
50%         2.040000
75%         2.050000
max         2.110000
Name: kw, dtype: float64
count    8.164055e+06
mean     1.716612e+22
std      4.904842e+25
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      3.070000e+00
max      1.401451e+29
Name: kw, dtype: float64
count    2.338586e+06
mean     5.992729e+22
std      9.164342e+25
min      5.000000e-03
25%      3.580000e+00
50%      4.790000e+00
75%      4.880000e+00
max      1.401451e+29
Name: kw, dtype: float64
```

In [6]:
```
# There seems to be a spike occuring around early to mid-February, it may be best to strip this o
active_df_post.plot(y='kw', figsize=(10,5), grid=True, title='kW Over Time', color='red', linesty
```

Out[6]:
```
<Axes: title={'center': 'kW Over Time'}, xlabel='time'>
```

In [8]:
```
# This aligns with a quirk that PowerTron indicated -- the device fluctuates and acts odd as thir
# Even several days after the treatment, the electricity usage seems much more in line with what
# after the treatment, things have continued to decline.

# Just stripping the first few days results in almost a 100% decrease in the mean value
# for kW given how wildly large the first results were.
df_post_clean1 = df_post[df_post['time'] > '2021-07-10 17:30:00']
clean_active_df_post1 = df_post_clean1[df_post_clean1['device_on'] == True]
clean_active_df_post1 = clean_active_df_post1.set_index(clean_active_df_post1['time'])
print(clean_active_df_post1['kw'].describe())

# The second stripping only results in a marginal decrease of 8%
# A worthwile method in the future may be to understand,
# when trimming the data, at what point the decreases may
# be attributed to seasonal fluctuations in device use vs any
# stabilization post treatment. 8% *seems* small
df_post_clean2 = df_post[df_post['time'] > '2021-10-01 17:30:00']
clean_active_df_post2 = df_post_clean2[df_post_clean2['device_on'] == True]
clean_active_df_post2 = clean_active_df_post2.set_index(clean_active_df_post2['time'])
print(clean_active_df_post2['kw'].describe())
```

```
count    2.192755e+06
mean     4.349618e+00
std      8.637420e-01
min      5.000000e-03
25%      3.570000e+00
50%      4.780000e+00
75%      4.870000e+00
max      1.182000e+01
Name: kw, dtype: float64
count    868450.000000
mean          3.984267
std           0.941081
min           0.005000
25%           2.980000
50%           4.550000
75%           4.760000
max          11.820000
Name: kw, dtype: float64
```

Aggregate Time Series + Training Sets + Weather Data

I was able to source some weather data, however, it is difficult to find weather data that contains multiple features measured minute by minute in a way that could be combined with the raw HVAC data. Iowa State University maintains the IEM (Iowa Environmental Mesonet) seemed to have the most complete dataset of weather-related data from the desired time period, which was available for download as a CSV file in one minute intervals [7]. This data was pulled from an archive of automated airport weather observations pulled from the Federal Aviation Administration's surface weather observation stations (ASOS/AWOS) [8]. The data available on the FIA website only allows for an individual to view the data for the past 120 hours, so fortunately Iowa State has developed a method of archiving this valuable data for download in CSV format.

Upon downloading the Fresno Air Terminal (FAT) data, it became very clear that the majority of feature measurements, beyond static items like the altitude of the site, were only taken in hourly increments. My goal with this data was to combine it with the original timeseries data, but given the variations in measurement frequency (minutes vs hours), I would need to resample the original time series data to combine the two. There are two options available for completing this step in the future. I could resample based on a built-in summary statistic, such as the mean. While this may appear to be a viable approach, many of the variables might produce non-representative summary statistics, such as the compressors that represent a binary of zero or one based on whether the device were on or not. Thus, it would make the most sense to strip the HVAC dataset down to only key variables that would be well represented by a summary such as the mean. Likely candidates for this include cfm, COP, eer, kw, kwton, totalCapacity_TONS, and totalCapacity_BTU. All of these features are derived from other measurements and are key indications of HVAC performance at any given moment, based on discussions with PowerTron.

So - I decided to take a shot at building one pre training set and one post training set as compilations of both the non-null weather data and the HVAC key features resampled as averages over a particular interval. Unfortunately, the resample produces a lot of null values, which I left and will discuss in further detail in the ML model discussion box below. Data cleaning and dealing with outliers is an element of data science and machine learning that I need to become more familiar with so I can navigate it confidently in the future. Below, after much learning, I got reasonably far into working to align the datasets based on their timestamps, but as I run out of time, I am still off by rows. In the case of the pre-treatment time period, I am only off by 1 row. In the case of the post-treatment time period, I am off by several hundred rows. I would

need to write a function that would allow me to further investigate where the rows depart from each other to better investigate this in the future.

In [9]:
```
# Import the weather data and take a look at where the temperature readings
# in degrees Farenheit are not null
weather_data = pd.read_csv('ASOS_weather.csv')
weather_subset = weather_data[weather_data['tmpf'].notnull()]
```

```
C:\Users\erine\AppData\Local\Temp\ipykernel_8936\2538483895.py:3: DtypeWarning: Columns (7,15,21,2
2,23,26) have mixed types. Specify dtype option on import or set low_memory=False.
  weather_data = pd.read_csv('ASOS_weather.csv')
```

In [10]:
```
# Clean up the weather data accordingly, dropping all columns showing null
# values which likely are due to lack of measurement recorded that could be
# pulled by the Iowa state algorithm, or lack of measurement altogether.

# This will still leave us with 9 features measured every minute

weather_subset['time'] = pd.to_datetime(weather_subset['valid'])
weather_subset.drop(['valid','station', 'gust', 'skyc1', 'skyc2', 'skyc3', 'skyc4', 'skyl1',
                     'skyl2', 'skyl3', 'skyl4', 'wxcodes', 'ice_accretion_1hr',
                      'ice_accretion_3hr', 'ice_accretion_6hr', 'peak_wind_gust',
                       'peak_wind_drct', 'peak_wind_time', 'metar', 'snowdepth'],
                       axis=1, inplace=True)
```

```
C:\Users\erine\AppData\Local\Temp\ipykernel_8936\610665416.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  weather_subset['time'] = pd.to_datetime(weather_subset['valid'])
C:\Users\erine\AppData\Local\Temp\ipykernel_8936\610665416.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  weather_subset.drop(['valid','station', 'gust', 'skyc1', 'skyc2', 'skyc3', 'skyc4', 'skyl1',
```

In [11]:
```
# Now take the weather data and subdivide it based on pre and post treatment
# timeframes
weather_subset_pre = weather_subset[weather_subset['time'] <= '2021-07-06 17:30:00']
weather_subset_pre['hourly_datetime'] = weather_subset_pre['time'].dt.strftime('%Y-%m-%d %H:00:0
weather_subset_pre = weather_subset_pre[(weather_subset_pre['hourly_datetime'] >= '2021-06-23 15
                                         (weather_subset_pre['hourly_datetime'] <= '2021-07-06 17

weather_subset_post = weather_subset[weather_subset['time'] >= '2021-07-06 17:30:00']
weather_subset_post['hourly_datetime'] = weather_subset_post['time'].dt.strftime('%Y-%m-%d %H:00
weather_subset_post = weather_subset_post[(weather_subset_post['hourly_datetime'] >= '2021-07-06
                                         (weather_subset_post['hourly_datetime'] <= '2022-02-08 0
```

```
C:\Users\erine\AppData\Local\Temp\ipykernel_8936\278920822.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  weather_subset_pre['hourly_datetime'] = weather_subset_pre['time'].dt.strftime('%Y-%m-%d %H:00:0
0')
C:\Users\erine\AppData\Local\Temp\ipykernel_8936\278920822.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/inde
xing.html#returning-a-view-versus-a-copy
  weather_subset_post['hourly_datetime'] = weather_subset_post['time'].dt.strftime('%Y-%m-%d %H:0
0:00')
```

In [12]:
```
# Ensure a subset of HVAC data starting on the same timeline as the weather
# data exists, so the timestamps align after subsetting the data and
# include only KPIs mentioned above

df_pre_Wsubset = df_pre[['time', 'cfm', 'COP', 'eer', 'kw', 'kwton',
                         'totalCapacity_TONS', 'totalCapacity_BTU']]
df_pre_Wsubset = df_pre_Wsubset[df_pre_Wsubset['time'] >= '2021-06-23 00:53:00']


df_post_Wsubset = df_post[['time', 'cfm', 'COP', 'eer', 'kw', 'kwton',
                         'totalCapacity_TONS', 'totalCapacity_BTU']]
df_post_Wsubset = df_post_Wsubset[df_post_Wsubset['time'] >= '2021-07-06 17:53:00']
```

In [13]:
```
# resample the HVAC dataframes above to only show hourly means for all values


# weather and hvac pre-treatment
df_pre_Wsubset = df_pre_Wsubset.set_index('time')
df_pre_Wsubset.index = pd.to_datetime(df_pre_Wsubset.index)
start_time_pre = df_pre_Wsubset.index[0].floor('H')
offset_pre = pd.Timedelta(df_pre_Wsubset.index[0] - start_time_pre)
hourly_hvac_pre = df_pre_Wsubset.resample('H', offset=offset_pre).mean()
hourly_hvac_pre = hourly_hvac_pre.reset_index()
hourly_hvac_pre['hourly_datetime'] = hourly_hvac_pre['time'].dt.strftime('%Y-%m-%d %H:00:00')

# weather and hvac post-treatment
df_post_Wsubset = df_post_Wsubset.set_index('time')
df_post_Wsubset.index = pd.to_datetime(df_post_Wsubset.index)
start_time_post = df_post_Wsubset.index[0].floor('H')
offset_post = pd.Timedelta(df_post_Wsubset.index[0] - start_time_post)
hourly_hvac_post = df_post_Wsubset.resample('H', offset=offset_post).mean()
hourly_hvac_post = hourly_hvac_post.reset_index()
hourly_hvac_post['hourly_datetime'] = hourly_hvac_post['time'].dt.strftime('%Y-%m-%d %H:00:00')
```

In [17]:
```
# By printing the pre and post-treatment datasets side by side, we get a better understanding of
# the overall understanding that the date/time indexing does not match, so we could not yet combi
# It thus becomes necessary to run additional tests and find where the indexing is off to delete
# searching was not completed at this time.

print("Weather Pre-Treatment Counts")
print(weather_subset_pre.count())
print("HVAC Pre-Treatment Counts")
```

```
print(hourly_hvac_pre.count())

print("Weather Post-Treatment Counts")
print(weather_subset_post.count())
print("HVAC Post-Treatment Counts")
print(hourly_hvac_post.count())
```

```
Weather Pre-Treatment Counts
tmpf               314
dwpf               314
relh               314
drct               304
sknt               313
p01i               314
alti               314
mslp               314
vsby               314
feel               314
time               314
hourly_datetime    314
dtype: int64
HVAC Pre-Treatment Counts
time                  315
cfm                    95
COP                    95
eer                    95
kw                     95
kwton                  95
totalCapacity_TONS     95
totalCapacity_BTU      95
hourly_datetime       315
dtype: int64
Weather Post-Treatment Counts
tmpf               5816
dwpf               5815
relh               5815
drct               5699
sknt               5811
p01i               5816
alti               5815
mslp               5150
vsby               5814
feel               5814
time               5816
hourly_datetime    5816
dtype: int64
HVAC Post-Treatment Counts
time                  5196
cfm                   4454
COP                   4454
eer                   4454
kw                    4454
kwton                 4454
totalCapacity_TONS    4454
totalCapacity_BTU     4454
hourly_datetime       5196
dtype: int64
```

Discussion on ML Models - A reflection and notes on Future Exploration

Intro

In all honestly, based on very limited experience I naively assumed that deploying some sort of supervised model would be as simple as downloading a library, plugging in a training set and an unlabeled dataset and letting it rip. I quickly found out that this was not the case. The biggest positives to my experience along this avenue are that I found several great resources online that will help me to learn machine learning at greater depth over the summer and I got to practice considering how certain features and assumptions may impact the outcome of a forecast.

I had planned to use a supervised technique whereby a model would be trained on some HVAC data such that it learns the relationships between features and then can predict what the kW rating might be at some point in the future. When I thought about it more, it made sense that I could not just train a model based on many relationships between the 37 HVAC features and then only give the model only two features (time and outside air temperature) and ask it to predict kWhs (based on a daily derivation) for some time in the future. The relationships it would be trained on are relational, so by not offering the model the same features in the unlabeled dataset, its predictions for the target variable would likely be skewed. This is the moment in which finding a secondary weather dataset became highly important. If the model could be trained on an aggregate data or even a dataset containing weather data with the added label of kW, it would be possible to pass the model the weather data for a period where HVAC data is unavailable. This strategy would render better results in terms of accuracy than my original starting assumption. So I began my research and made an attempt described below under "Approach I" and hit a wall. I then completed more research and discovered the methods cited in "Future Approaches," which I hope to try in the future when given more time.

Beyond the approaches below, it came to me towards the end of the project that the reliance on weather data as the features for the supervised model may still produce inaccurate results in this scenario. While assuming a relationship between weather and kWh (electricity consumption) makes sense in the case of residential or commercial circumstances where the HVAC is being used to cool humans in reaction to the rising temperatures, the seasonality may be less when the HVAC device is being used to cool mechanical equipment, whose heating and cooling needs are less strongly driven by outdoor weather conditions, though still weakly related. In the case of my data, the cooling needs of equipment at a cell site in Fresno may be related to weather, as when the temperature outdoors increases, the more quickly the equipment heats up and the harder it is to cool it down. However, there may be scenarios when the temperature is cool and still the equipment requires much cooling as it received a difficult job or task that heats things up. This approach may not be the best in producing simulated data for an entire year and other approaches should be considered and researched beyond this paper.

Approach I

With the time given, I made a single approach using Microsoft's LightGBM, which is a gradient boost regression model I had seen used in an example of time series forecasting completed by Unai López Ansoleaga [9]. This approach can be expanded to create a multi-output regressor using decision trees [10] I started working to set this model up and understand it in tandem with the cleaning of the data above. Things quickly became messy as Ansoleaga's description is written for an audience that already has LightGBM up and running or can complete that part themselves.

I had previously worked with unsupervised models like k-means clustering and hierarchical clustering. These models are well-known and widely used, so it is easy to download and deploy a python library thaat allows you to deploy the model on your data. In the case of LightGBM, there are no Python libraries or easy downloads. The installation documentation is riddled with missing information and contains extremely little on troubleshooting and debugging during the installation. Furthermore, the installation guide itself is written for someone who has built a model in visual studio before. The learning curve to deploying something like this is steep and with debugging, I spent ~5-6 hours on working to debug and trying to get the model up in visual studio and importing it to Python from there. My attempts were unsuccessful and I became frustrated quickly.

While Approach I was not a technical success, it humbled me and unveiled an area in which I wish to learn more and improve. Google's machine learning course infrastructure covers gradient boosting and gradient boosted decision trees under "advanced courses", which would point to the conclusion that I may have been jumping the gun a little bit in trying to deploy this model [11]. I took this in stride and understand now that I may need more foundational work and understanding before diving in the deep end and deploying certain machine learning techniques and models - greater funcitonal knowledge.

The approach did lead me to completing some basic time series decomposition and statistical manipulations in preparation for deploying the model, shown in the appendix. I completed some additional research on such manipulation and have included several concepts I wish to explore further, as they likely will boost future explorations and manipulations of time series prior to pulling out the big/fancy computational guns of machine learning models. It is worthwhile to understand the statistical manipulations as they may be more efficient and produce comparable results.

Future Approaches

When I became frustrated with LightGBM and also considered the limitations of having a supervised model that spits out only one HVAC function, kW, I stumbled across another area of advanced machine learning: multi-output regression and multi-step multi-output models. These models would allow for a model to be trained such that it learns the relationships between multiple inputs and multiple labeled outputs and can produce a set of targets/forecasted values for some point in the future. This set-up would allow me to use the weather data and HVAC behavior to produce multiple HVAC features (see the discussion of KPIs under "Aggregate Time Series + Training Sets + Weather Data") based on multiple features of weather data.

I discovered several approaches in this vein that I wish to explore in the future and read about, but due to time constraints, did not implement:

- Using neural networks and deep learning for multi-output regression [12][13]
- Utility of various types of regressors, such as k-neighbors, decision trees, and random forest regressors [10]
- Multi-output regression as a tactic [14]
- Long Short-term Memory Networks for Multi-Step Multi-Output prediction (use of recurrent neural netwoks) [15]

I am thrilled by the opportunity to have so much to learn and real data that I can continue to wrangle to test out some of these models while learning them as opposed to just reading about them. I also believe it was the appropriate choice to avoid jumping on one article to duplicate the approach, as I learned this can be fruitless and frustrating based on the approach described in section I, "Approach I". This experience

reaffirmed my belief that while it may be tempting and flashy to try and deploy these models if you know a bit of coding and have a computer, to truly deploy these in a meaningful way, or to leverage advanced approaches most appropriate to your problem, it is best to understand what kind of black box you might be building and shoving your data into, or you have little chance of catching limitations or bias in the output.

Conclusion + Reflection

I began this project with the expectation that the above research and data cleaning/manipulation would not constitute a large time investment. This was by far my biggest area of underestimation and a cautionary tale to be reminded of in future working projects when estimating timelines. Until you have really wrestled with the data, you have little understanding of what anomalies and null values may be lurking to throw a wrench in things. For this project, I chose to work with new site data, as the previous work I completed with PowerTron was centered around what I now know to be a luxurious dataset from a big box store. The cell site data was a new beast entirely -- not all sensors fit all machines equally it seems. The device function itself on the cell site used behaved in a sporadic fashion in comparison to the routine use of an HVAC placed in a commercial office space. The fluctuations seen in the days following the treatment appeared to have much stronger effects than those in the previous dataset I had handled.

I feel glad to have selected a dataset that was more difficult to handle, as from what I have heard, data wrangling is a lot of what you do in a data science job in the real world. I achieved some excellent practice and learned areas for possible improvement as follows:

- working to better understand how to build an ML model that is not simply a downloadable library or something with a polished .exe file (LightGBM) and understand gradient boost frameworks in general
- gaining deeper understanding of various ML algorithms and their functionality.
- working to increase understanding of industry best practices when handling outliers and messy data: what questions should one ask about the data? when is it ok to toss null values? etc.
- improving skills in method chaining in pandas
- understanding how to deploy lambda functions to complete complex conditional operations when subsetting data in pandas
- exploring models for producing multi-output forecasts in greater detail
- better understanding basic statistics so as to understand when a more efficient/functional model works as well if not better than computational approaches to time series forecasting. This will also ensure that frivolous models aren't deployed when they aren't needed which can result in frivilous computation and thus frivolous energy use that can add up, as seen in course readings on Sustainable Computation. While maximizing efficiency in algorithms themselves is the useful approach of green AI, an even more fruitful approach is ensuring complex models are not deployed due to flashiness and popularity when tried and true methods are equally capable and most efficient [16]
- diving deeper into better understanding electricity pricing and utility finance; in a capitalist economy, this is a key space to understand. Pricing, financing and the relationships of utilities/grids/generators etc. are not only what shape the priorities and intentions of players in the space, but cost and benefit quantified in real dollars make powerful incentives that help to change behavior in an energy transition.

References

[1] - PG&E. "Electric Service Area Maps." Pacific Gas and Electric. pge.com. Dec. 17th 2014. https://www.pge.com/tariffs/assets/pdf/tariffbook/ELEC_MAPS_Service%20Area%20Map.pdf

[2] - US Energy Information Administration. "Glossary - K - kWh." eia.gov. May 10th 2023. https://www.eia.gov/tools/glossary/index.php?id=K#kwhour

[3] - PG&E. "Tariffs." Pacific Gas & Electric. pge.com. May 10th 2023. https://www.pge.com/tariffs/index.page

[4] - PG&E. "Tariffs - Advice Letter Index (2021)" Pacific Gas & Electric. pge.com. Dec 31st 2021. https://www.pge.com/tariffs/advice-filing-index.page?xmldoc=sites-data/tariffs/data/advice-letters/2020/electric.xml

[5] - Meadows, Donella. "Leverage Points: Places to Intervene in a System.Links to an external site." 1999. https://donellameadows.org/wp-content/userfiles/Leverage_Points.pdf

[6] - Marsh, Jacob. "How Many Watts Does an Air Conditioner User?" energysage.com. Jan 11th 2022. https://news.energysage.com/how-many-watts-does-an-air-conditioner-use/#:~:text=On%20average%2C%20a%20central%20AC%20system%20uses%203%2C000%20to%204%2C000%

[7] Iowa Environmental Mesonet (IEM). "ASOS-AWOS-METAR Data Download." Iowa State University. 2021. https://mesonet.agron.iastate.edu/request/download.phtml?network=CA_ASOS

[8] "Surface Weather Observation Stations (ASOS/AWOS)." Federal Aviation Administration. May 10th 2023. https://www.faa.gov/air_traffic/weather/asos

[9] Ansoleaga, Unai Lopez. "Time Series Forecasting with Supervised Machine Learning." towardsdatascience.com. Medium. Jan 5th 2022. https://towardsdatascience.com/time-series-forecasting-with-machine-learning-b3072a5b44ba

[10] Brownlee, Jason. "How to Develop Multi-Output Regression Models with Python." machinelearningmastery.com. Guiding Tech Media. Mar 27th 2020. https://machinelearningmastery.com/multi-output-regression-models-with-python/

[11] "Machine-Learning - Decision Forests." developers.google.com. Google. May 10th 2023. https://developers.google.com/machine-learning/decision-forests

[12] Follonier, Florian. "Stock Market Forecasting Neural Networks for Multi-Output Regression in Python" relataly.com. Jul 13th 2021. https://www.relataly.com/stock-price-prediction-multi-output-regression-using-neural-networks-in-python/5800/

[13] Brownlee, Jason. "Deep Learning Models for Multi-Output Regression." machinelearningmastery.com. Guiding Tech Media. Aug 28th 2020 https://machinelearningmastery.com/deep-learning-models-for-multi-output-regression/

[14] Randolph, Cory. "Explainable AI for Multi-Output Regression." towardsdatascience.com. Medium. Feb 26th 2021. https://towardsdatascience.com/explainable-ai-for-multiple-regression-2df70cfc9995

[15] Nicapotato. "Keras Timeseries Multi-Step Multi-Output." kaggle.com. Google. 2020 https://www.kaggle.com/code/nicapotato/keras-timeseries-multi-step-multi-output

[16] Schwartz, Roy, et al. "Green AI." Communications of the ACM 63.12 (2020): 54-63. https://dl.acm.org/doi/pdf/10.1145/3381831

[17] Peixeiro, Marco. "The Complete Guide to Time Series Analysis and Forecasting." towardsdatascience.com. Medium. Aug 7 2019. https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

[18] Wobser, Greg. "Practical Methods for Identifying Seasonality in a Dataset." demand-planning.com. Institute of Business Forecasting & Planning. Feb 24 2021. https://demand-planning.com/2021/02/24/practical-methods-for-identifying-seasonality-in-a-dataset/

[19] Etienne, Benjamin. "Time Series in Python — Part 2: Dealing with seasonal data." towardsdatascience.com. Medium. Feb 15th, 2019. https://towardsdatascience.com/time-series-in-python-part-2-dealing-with-seasonal-data-397a65b74051

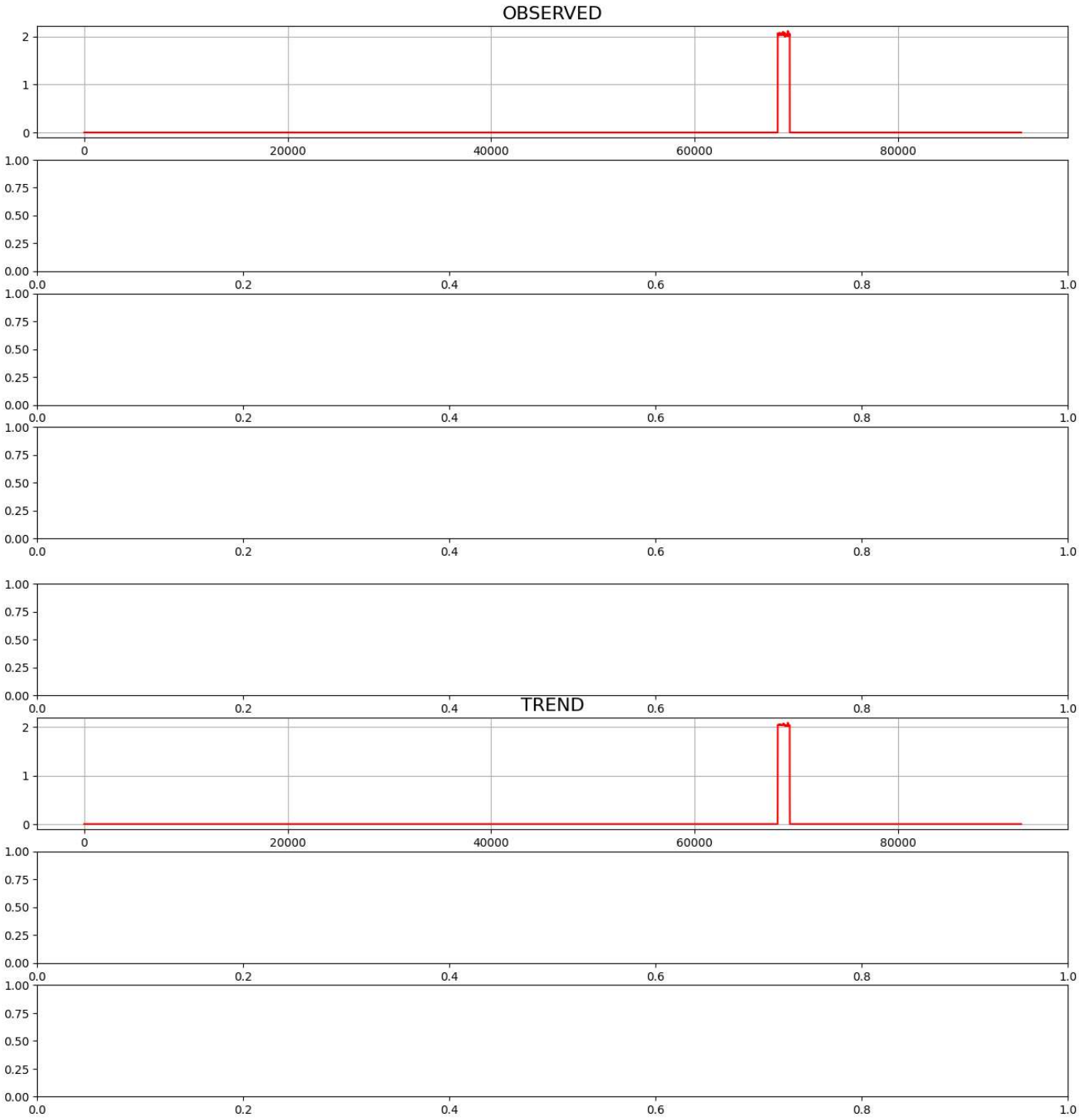
Appendix - Experiments on Seasonality

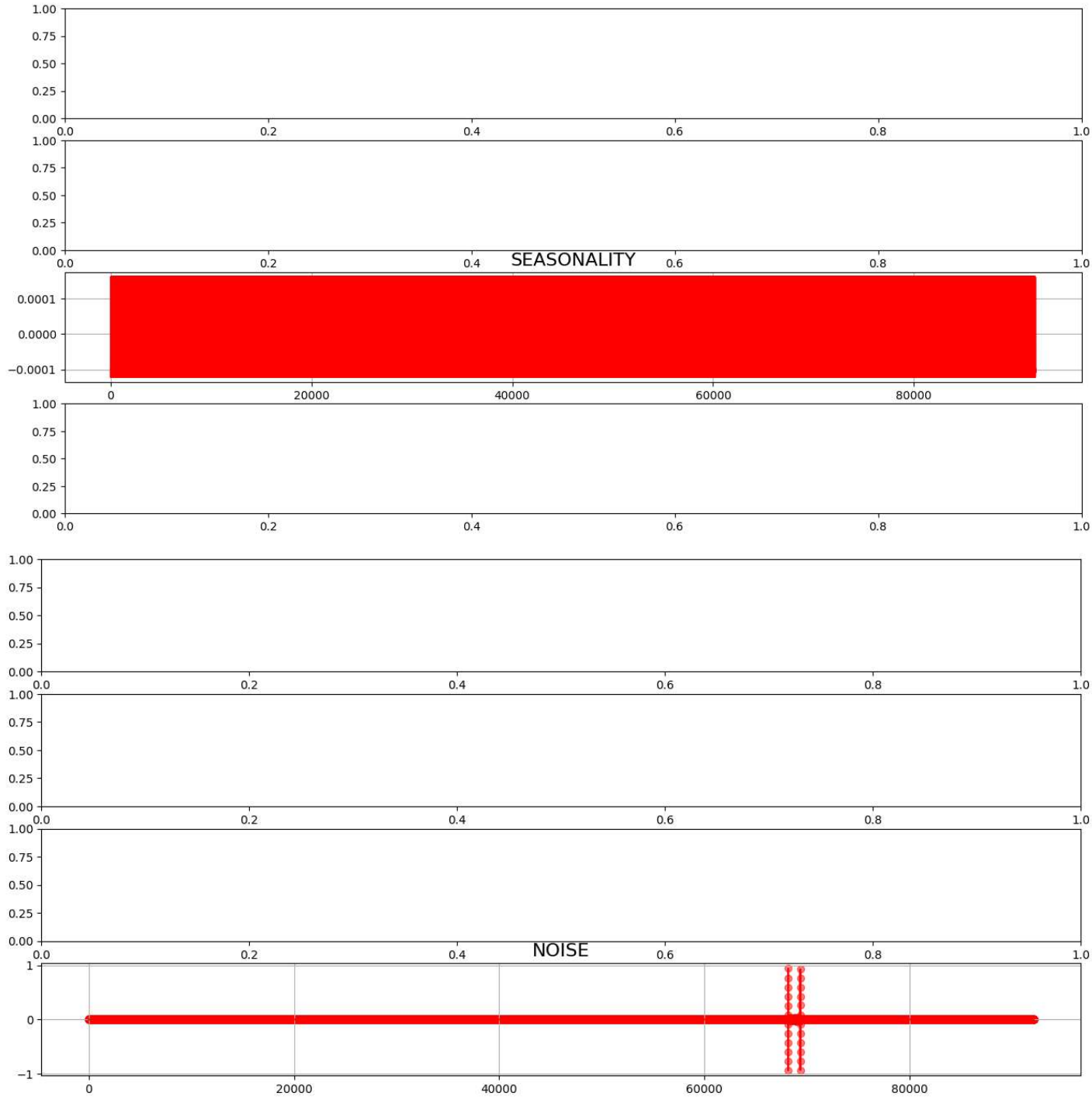
The below cells I produced as part of the prep work for "Approach I" described in the "Discussion on ML Models" section above. I wanted to retain the code and also show, based on the autocorrelation plot, that the data displays a ton of seasonality. This was also an indication that I want to learn more about producing these useful statistics plots and visualizations in a manner that is both efficient an legible for large datasets in the future [17][18][19].

In [25]:
```
# Decompose the time series to take a look at seasonality and noise
def decompose_kw_consumption(df, share_type='kw', samples=250, period=24):
    if samples == 'all':
        #decompose all time series timestamps
        res = seasonal_decompose(df[share_type].values, period=period)
    else:
        #decomposing a sample of the time series
        res = seasonal_decompose(df[share_type].values[-samples:], period=period)

    observed = res.observed
    trend = res.trend
    seasonal = res.seasonal
    residual = res.resid

    colors = ['red', 'green', 'blue', 'orange', 'purple']
    plt.rcParams['axes.prop_cycle'] = plt.cycler(color=colors)

    #plot the complete time series
    fig, axs = plt.subplots(4, figsize=(16,8))
    axs[0].set_title('OBSERVED', fontsize=16)
    axs[0].plot(observed)
    axs[0].grid()

    fig, axs = plt.subplots(4, figsize=(16,8))
    axs[1].set_title('TREND', fontsize=16)
    axs[1].plot(trend)
    axs[1].grid()

    fig, axs = plt.subplots(4, figsize=(16,8))
    axs[2].set_title('SEASONALITY', fontsize=16)
    axs[2].plot(seasonal)
    axs[2].grid()

    fig, axs = plt.subplots(4, figsize=(16,8))
```

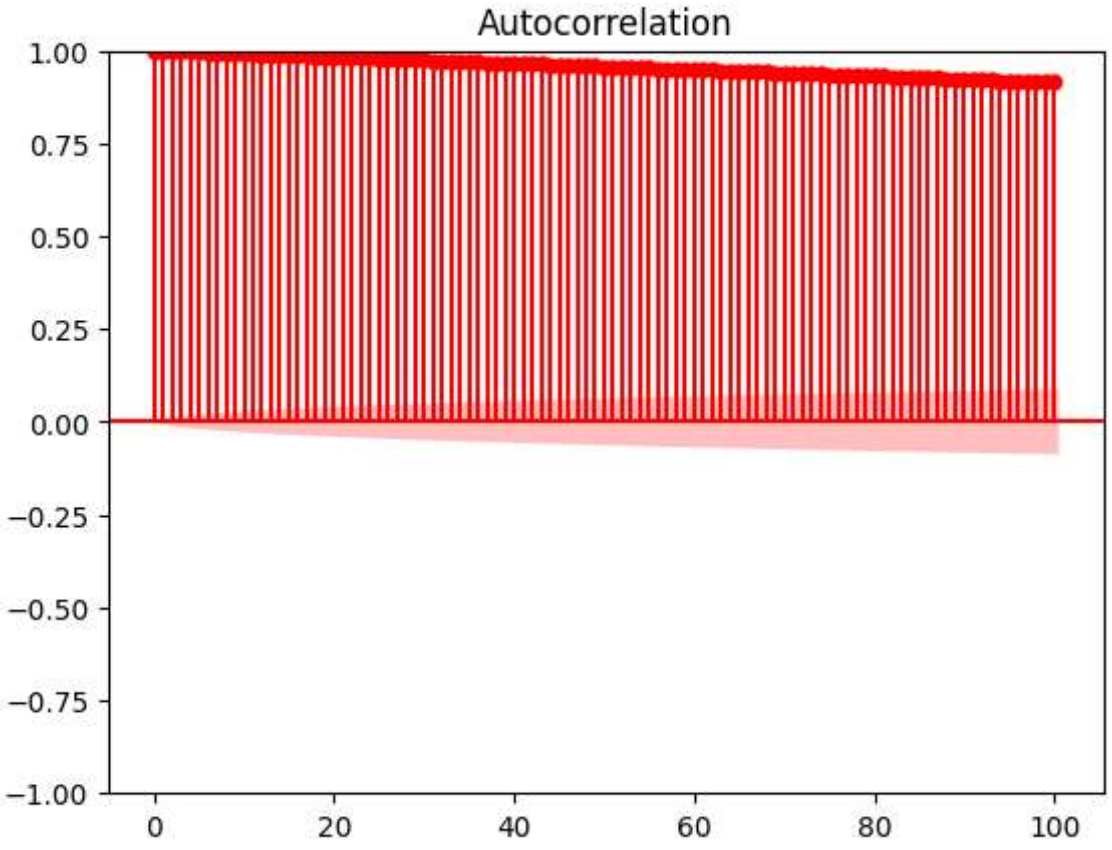
```
    axs[3].set_title('NOISE', fontsize=16)
    axs[3].plot(residual)
    axs[3].scatter(y=residual, x=range(len(residual)), alpha=0.5)
    axs[3].grid()

    plt.show()
```

In [26]:
```
decompose_kw_consumption(df_pre, samples='all', period=12)
```





In [20]:
```
plot_acf(df_pre['kw'].values, lags=100)
plt.show()
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100